Trade-offs for Threshold Implementations Illustrated on AES

Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen

***Abstract*—Embedded cryptographic devices are vulnerable to power analysis attacks. Threshold Implementations provide provable security against first-order power analysis attacks for hardware and software implementations. Like masking, the approach relies on secret sharing but it differs in the implementation of logic functions. While masking can fail to provide protection due to glitches in the circuit, Threshold Implementations rely on few assumptions about the hardware and are fully compatible with standard design flows. We investigate two important properties of Threshold Implementations in detail and point out interesting trade-offs between circuit area and randomness requirements. We propose two new Threshold Implementations of AES that, starting from a common previously published implementation, illustrate possible trade-offs. We provide concrete ASIC implementation results for all three designs using the same library, and we evaluate the practical security of all three designs on the same FPGA platform. Our analysis allows us to directly compare the security provided by the different trade-offs, and to quantify the associated hardware cost.**

***Keywords*—Threshold Implementation, First-order DPA, Higher-order DPA, Glitches, Sharing, AES, S-box**

I. Introduction

An increasing number of embedded devices implement some security functionality, for instance smart cards (banking, SIM, public transport, access control, passports), car keys, set-top boxes (pay TV), media players, mobile phones, tablets, medical implants, etc. These devices use cryptographic algorithms that are secure against mathematical cryptanalysis. This means that a system's security relies on the secrecy of a so-called cryptographic key, and that there are no mathematical shortcuts that allow to break the system. However, in the late 90s the security of such devices has been shown to depend also on the algorithms' implementation [1]. During the computation of an algorithm the device leaks information, for instance through its power consumption, electromagnetic emanations, etc. Side channel attacks (SCA) can reveal the key from these leakages and are often inexpensive, hence they are among the most relevant threats for the security of implementations of cryptographic algorithms. Certain countermeasures against SCA aim to introduce noise in the side channel, e.g. random delays, random order execution, dummy operations, etc., while masking conceals all sensitive intermediate values of a computation with random data. Different masking schemes, like additive [2], [3] and multiplicative [4], have been proposed in order to provide security against differential power analysis (DPA) attacks. In d^{th}-order additive masking, each sensitive intermediate value x of the algorithm is represented and processed in $d+1$ shares $x_1, \ldots, x_d, x_{d+1}$ where the first d shares are chosen at random and the last share is chosen such that $x_1 \oplus \ldots x_d \oplus x_{d+1} = x$. Masking allows one to formally argue the security it provides against DPA.

However, it was shown [5]–[7] that masked hardware implementations can still be vulnerable to first-order DPA due to the presence of glitches. One can try to eliminate the security relevant glitches by carefully balancing signal propagation delays or by using special "secure" logic styles, but this is not always compatible with standard design flows, is poorly supported by standard tools, requires expertise, time, iterations of design and testing, and hence is expensive. As an alternative, new masking schemes have been developed that provide provable security for a circuit generated with a standard design flow even if glitches occur in the circuit.

Related Work: In 2006 Nikova et al. proposed such a scheme called Threshold Implementation (TI) [8]. It is based on secret-sharing and provides provable security against first-order DPA [9]. In 2011 Prouff and Roche proposed a d^{th}-order masking scheme [10], based on Shamir's secret sharing, for which they claim security even against higher-order attacks. It is a general method that replaces every field multiplication by $4d^3$ field multiplications and $4d^3$ additions, using $2d^2$ bytes of randomness. For resource constrained embedded applications this may prove too costly or inefficient. Moreover, it has been shown in [11] that straightforward implementations of this scheme may not be secure in practice.

The TI technique is based on a specific type of multi-party computation and applies Boolean masking. Interesting properties of the technique are that it provides provable security against first-order side-channel attacks, that it requires few assumptions on the hardware leakage behavior, that it is fully compatible with standard CMOS and FPGA design flows, and that it allows to construct realistic-size circuits without intervention and design iterations. However, TIs can still be broken by univariate mutual information analysis (MIA) [9], [12] or univariate higher-order attacks [13].

It has been shown that all 3×3 and 4×4 S-boxes have a TI sharing with 3, 4 or 5 shares [14]. The TI approach has

This work has been supported in part by the Research Council of KU Leuven (OT/13/071 and GOA/11/007), by the FWO (G.0550.12) and by the Hercules foundation (AKUL/11/19). B. Bilgin, B. Gierlichs, S. Nikova, and V. Rijmen are affiliated with KU Leuven, ESAT-COSIC and iMinds, Belgium, {name.surname}@esat.kuleuven.be. B. Bilgin is also affiliated with University of Twente, EEMCS-SCS, The Netherlands and partially supported by the FWO project G0B4213N. B. Gierlichs is a Postdoctoral Fellow of the Research Foundation - Flanders (FWO). V. Nikov is affiliated with NXP Semiconductors, Belgium.

Copyright (c) 2015 IEEE. Personal use of this material is permitted.
However, permission to use this material for any other purposes must be obtained from the IEEE by sending an email to pubs-permissions@ieee.org

been applied to only a few entire algorithms: PRESENT [15], AES [16], [17], KECCAK [18] and Fides [19]. In AES, the S-box is by far the most challenging part to share. Moradi et al. [16] proposed a TI of this S-box, based on the tower field approach [20], that constantly uses three shares. They also use three shares for the AES implementation. In contrast, Bilgin et al. [17] proposed to change the number of shares for different stages of the tower field approach, and for the AES implementation.

Contribution: This paper is an extended version of our paper at AfricaCrypt 2014 [17]. We proposed a TI of AES-128 encryption that requires about 9k GE with the library that we use and 44 bits of additional randomness per S-box calculation. We used the tower field approach over GF(2^4) for the S-box and we adapted the number of shares for each function in the S-box computation to minimize the overall gate count of the S-box. We used only two shares for most of the linear operations and hence had two sets of registers for state update and key schedule. All functions were uniformly shared and the number of shares went up to five in the S-box. We used re-masking to satisfy the uniformity in the whole circuit when the uniformly shared functions are combined. Our practical security evaluation confirmed the expected first-order DPA resistance and identified the linear part in two shares as the most vulnerable part of the implementation.

In this extended version, we investigate the uniformity problem and the need for re-masking in more detail. We prove that under certain circumstances, it is enough to re-mask only a fraction of the shares. Moreover, we argue that if there is enough re-masking, we do not need to share functions uniformly. This observation helps us to further reduce the area and randomness requirements. We provide two new implementations. The first one is similar to the one in [17], but it uses at least three shares in all the operations, including the linear ones. We use it to investigate the increase in security when moving from at least two to at least three shares, and to quantify the associated cost. The second implementation is based on the one in [17] but modified according to our findings regarding uniformity and re-masking. It requires only about 8k GE with the library that we use and 32 bits of additional randomness per S-box calculation. Our three implementations need the same number of clock cycles to complete the calculation, and allow us therefore to focus on some trade-offs between area and additional randomness. Moreover, we provide results of practical security evaluations of all three implementations on the *same* FPGA platform and under the *same* lab conditions. They confirm the theoretically guaranteed first-order attack resistance for all implementations and allow us to complement the study of trade-offs with an analysis of our implementations' security against higher-order attacks.

II. THRESHOLD IMPLEMENTATIONS

We recall and clarify the definitions and security theorems of Threshold Implementations.

A. Notation and Definitions

Lower-case characters refer to elements of a finite field and functions over finite fields, while upper-case characters are used for stochastic variables. We denote a vector or a vector function with bold characters. Let $X \in \mathcal{F}^m$ denote the input of the (unshared) function f. A *masking* (share vector) $\mathbf{X}$ of X is the result of a stochastic function that takes as inputs a value x and some auxiliary values (*random masks*), and that outputs a vector $\mathbf{x}$ containing shares $x_1, x_2, \ldots, x_{s_x}$ such that the XOR-sum of the s_x shares equals x. For all values x with $\Pr(X = x) > 0$, let $\mathrm{Sh}(x)$ denote the set of valid share vectors $\mathbf{x}$ for x:

$$\mathrm{Sh}(x) = \{\mathbf{x} \in \mathcal{F}^{ms_x} \mid x_1 + x_2 + \cdots + x_{s_x} = x\} .$$

$\Pr(\mathbf{X} = \mathbf{x}|X = x)$ denotes the probability that $\mathbf{X} = \mathbf{x}$ when the unshared input of the masking equals x, taken over all auxiliary inputs of the masking. Similarly, we denote the output $Y \in \mathcal{F}^n$, and corresponding $s_y, \mathbf{y}$ and $\mathrm{Sh}(y)$. Let $\mathbf{f}$ denote the vector function composed of the component functions $f_1, \ldots, f_{s_y}$ operating on the share vector $\mathbf{x}$ and outputting $\mathbf{y}$; we will call it a *sharing* of the function.

The scheme, like most other masking schemes, requires that the masking is *uniform*, in the sense of the following definition.

Definition 1 (Uniform masking): A masking $\mathbf{X}$ is *uniform* if and only if there exists a constant p such that for all x we have:

$$\begin{aligned} &\text{if } \mathbf{x} \in \mathrm{Sh}(x) \text{ then } \Pr(\mathbf{X} = \mathbf{x}|X = x) = p, \\ &\text{else } \Pr(\mathbf{X} = \mathbf{x}|X = x) = 0 \end{aligned}$$

and

$$\sum_{\mathbf{x} \in Sh(x)} \Pr(\mathbf{X} = \mathbf{x}) = \Pr(X = x) .$$

In words, we call a masking uniform if for each value x of the variable X, the corresponding vectors with masked values occur with the same probability.

Threshold Implementations use sharings that satisfy *correctness* and *non-completeness* properties which are defined as follows.

Definition 2 (Correctness): The sharing $\mathbf{f}$ is *correct* if and only if $\forall x \in \mathcal{F}^m$, $\forall y \in \mathcal{F}^n$:

$$\forall \mathbf{x} \in \mathrm{Sh}(x), \forall \mathbf{y} \in \mathrm{Sh}(y); \mathbf{f}(\mathbf{x}) = \mathbf{y} \Leftrightarrow f(x) = y.$$

Definition 3 (Non-completeness): A sharing $\mathbf{f}$ is *non-complete* if every component function of $\mathbf{f}$ is independent of at least one share x_i of $\mathbf{x}$.

B. Security Proofs

We start with a lemma proving that uniformity of a masking implies the independence that we need for the proof of Theorem 1. Let $\mathbf{X}_{\bar{i}}$ denote the vector obtained by removing X_i from $\mathbf{X}$.

Lemma 1: If the masking $\mathbf{X}$ of X is uniform, then $\mathbf{X}_{\bar{i}}$ and X are independent (for any choice of i).

Proof: Two stochastic functions are independent if and only if their joint distribution equals the product of their marginal distributions. Hence, we have to show for all i that

$$\forall \mathbf{x}_{\bar{i}}, x : \Pr(X = x, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}) = \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}) \Pr(X = x).$$

Since $\Pr(A,B) = \Pr(B)\Pr(A|B)$, it suffices to show that $\forall \mathbf{x}_{\bar{i}}, x : \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}} | X = x) = \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}})$. We start from

$$\begin{aligned}\Pr(\mathbf{X} = \mathbf{x}|X = x) &= \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}, X_i = x_i | X = x)\\ &= \frac{\Pr(X = x, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}, X_i = x_i)}{\Pr(X = x)}\\ &= \frac{\Pr(X = x, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}, X_i = x_i)}{\Pr(X = x, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}})} \frac{\Pr(X = x, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}})}{\Pr(X = x)}\\ &= \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}} | X = x) \Pr(X_i = x_i | X = x, \mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}) .\end{aligned}$$

We know that the last factor equals 1 when $\mathbf{x} \in \mathrm{Sh}(x)$ and zero otherwise. Hence we obtain

$$\forall x : \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}} | X = x) = p. \tag{1}$$

Now we can write (Bayes' Theorem):

$$\begin{aligned}\Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}}) &= \sum_x \Pr(\mathbf{X}_{\bar{i}} = \mathbf{x}_{\bar{i}} | X = x) \Pr(X = x)\\ &= p \sum_x \Pr(X = x) = p. \end{aligned} \tag{2}$$

The equality of Eqs. (1) and (2) proves the claim. ■
It follows that $p = |\mathcal{F}|^{m(1-s_x)}$.

The security against first-order side-channel attacks in circuits satisfying correctness and non-completeness follows now from two intuitively easy steps. We start from a result on the individual component functions.

Theorem 1 ([9]): If the masking $\mathbf{X}$ is uniform and the shared function $\mathbf{f}$ (hence the circuit of $\mathbf{f}$) is non-complete, then any single component function of $\mathbf{f}$ does not leak any information on X.

The proof of this theorem is simple and intuitive (see [9] for the formal proof). Every component function works on an input $\mathbf{X}_{\bar{i}}$ for some i. Lemma 1 states that such an input is independent of X. In other words, a component function does not get the information to determine the value of X. Since the function does not *know* x, it cannot leak x. Note that we do not have to make *any* assumption on the physical behavior of the hardware or software implementation of the *component functions*.

Finally, we look at the whole circuit. Even though the component functions of $\mathbf{f}$ can be made independent of X individually, we cannot achieve independence for the whole circuit. However, due to the linearity of the expectation operator, we can still prove independence of the average value and therefore resistance against first-order attacks. Let $\mathcal{L}$ denote a leakage signal of an implementation of the circuit $\mathbf{f}$, be it instantaneous or summed over an arbitrary period of time. We require that the leakage of the whole circuit is the sum of the leakages of the sub-circuits.

Theorem 2 ([9]): If the masking $\mathbf{X}$ is uniform and the circuit of $\mathbf{f}$ is non-complete, then the expected value (average) of $\mathcal{L}$ is constant.

The proof uses only elementary probability theory. Due to the linearity of the expectation operator, the expected value of $\mathcal{L}$ is the sum of the expected values of the leakages of the component functions. Theorem 1 implies that the expected values of the leakages of the component functions are constant. Hence, so is the expected value of $\mathcal{L}$.

Note that the only required assumption on the physical behavior of the hardware or software implementation of $\mathbf{f}$ is that the component functions can be implemented such that the leakage from each of them is independent of at least one share of X. In other words, the cross-talk between implementations of different components should be negligible. However, the theorem claims results only on the expected value of $\mathcal{L}$, since higher-order statistical moments are not linear.

C. What Can Go Wrong without Uniformity?

We show by means of a simple example what can go wrong if a sharing is not uniform. Note that the non-uniform sharing of the 5×5 S-box of the SHA-3 competition winner Keccak [18], [21] has similar problems. Let $(A, B) \in \mathbb{F}_2^2$ and their product $\mathbb{F}^2 \ni C = f(A, B) = AB$. Define $\mathbf{f}$ as follows:

$$\begin{aligned} C_1 &= f_1(A_2, A_3, B_2, B_3) = A_2B_2 + A_2B_3 + A_3B_2\\ C_2 &= f_2(A_1, A_3, B_1, B_3) = A_3B_3 + A_1B_3 + A_3B_1\\ C_3 &= f_3(A_1, A_2, B_1, B_2) = A_1B_1 + A_1B_2 + A_2B_1 . \end{aligned} \tag{3}$$

If the masking of the input (A, B) is uniform, then the masking of C is distributed as shown in Table I. In order to satisfy the uniformity of the masking of the output C, we would need that the 16 non-zero values in the table were equal (specifically to $2^{2(3-1)-1(3-1)} = 4$ as will be defined in Definition 4). Theorem 2 implies that there is no leakage of information in *this* circuit. However, if C is used as input of a second circuit, then Theorem 2 does not apply anymore to the second circuit (because its inputs are not uniform) and potentially the second circuit might leak information.

Table I: Number of times that a masking c_1, c_2, c_3 occurs for a given input (a, b).

	c_1, c_2, c_3							
(a, b)	000	011	101	110	001	010	100	111
$(0, 0)$	7	3	3	3	0	0	0	0
$(0, 1)$	7	3	3	3	0	0	0	0
$(1, 0)$	7	3	3	3	0	0	0	0
$(1, 1)$	0	0	0	0	5	5	5	1

Let $E = D \times C$ and let this multiplication be implemented by similar formulas as above. For example, Eq. (3) becomes:

$$E_1 = f_1(C_2, C_3, D_2, D_3) = C_2D_2 + C_2D_3 + C_3D_2 . \tag{4}$$

Assume that the masking of D is uniform but the masking of C has the distribution given in Table I. Then the masking of E will be distributed as shown in Table II. The average Hamming weight of E_1, E_2, E_3 in the seventh row ($(a, b, d) = (1, 1, 0)$) equals $33/32$, whereas it equals $27/32$ in the first six rows. This implies that some hardware implementations might show a different average power consumption when $(a, b, d) = (1, 1, 0)$. Observe also that the correlation between C_i and C is 0.125. Hence, in the part of the circuit implementing Eq. (4), the average of the leakage $\mathcal{L}$ can be correlated to C, since both C_2 and C_3 are correlated to C.

Table II: Number of times that a masking e_1, e_2, e_3 occurs for a given input (a, b, d).

	e_1, e_2, e_3							
(a, b, d)	000	011	101	110	001	010	100	111
$(0, 0, 0)$	37	9	9	9	0	0	0	0
$(0, 0, 1)$	37	9	9	9	0	0	0	0
$(0, 1, 0)$	37	9	9	9	0	0	0	0
$(0, 1, 1)$	37	9	9	9	0	0	0	0
$(1, 0, 0)$	37	9	9	9	0	0	0	0
$(1, 0, 1)$	37	9	9	9	0	0	0	0
$(1, 1, 0)$	31	11	11	11	0	0	0	0
$(1, 1, 1)$	0	0	0	0	21	21	21	1

D. Uniformity as a Remedy

We can take different types of actions to remedy the problem described in the previous section. First, we can apply *re-masking* as done by Moradi et al. [16]: by adding fresh masks to the shares C_1, C_2, C_3, we make the distribution uniform. A discussion on re-masking is provided in Section II-F. Second, we can impose an extra condition on $\mathbf{f}$ such that the distribution of its output is always uniform. This extra condition is the uniformity defined below.

Definition 4 (Uniform sharing of a function (circuit)): The sharing $\mathbf{f}$ is *uniform* if and only if

$$\forall x \in \mathcal{F}^m, \forall y \in \mathcal{F}^n \text{ with } f(x) = y, \forall \mathbf{y} \in \mathrm{Sh}(y) : \\ |\{\mathbf{x} \in \mathrm{Sh}(x) | \mathbf{f}(\mathbf{x}) = \mathbf{y}\}| = \frac{|\mathcal{F}|^{m(s_x-1)}}{|\mathcal{F}|^{n(s_y-1)}} \ .$$

If $s_x = s_y$ and $m = n$, this simplifies to:

$$\forall x, y \in \mathcal{F}^m \text{ with } f(x) = y, \forall \mathbf{y} \in \mathrm{Sh}(y) : \\ |\{\mathbf{x} \in \mathrm{Sh}(x) | \mathbf{f}(\mathbf{x}) = \mathbf{y}\}| = 1 \ .$$

It follows that a uniform circuit $\mathbf{f}$ is invertible if and only if f is invertible. We now prove that the uniform circuit condition is sufficient to achieve a uniform distribution at its output.

Theorem 3: If the masking $\mathbf{X}$ is uniform and the circuit $\mathbf{f}$ is uniform, then the masking $\mathbf{Y}$ of $Y = f(X)$, defined by $\mathbf{Y} = \mathbf{f}(\mathbf{X})$ is uniform.

Proof: In order to prove that $\mathbf{Y}$ is uniform, we need to show that $\Pr(\mathbf{Y} = \mathbf{y} | Y = y)$ is equal to a constant p if $\mathbf{y} \in \mathrm{Sh}(y)$ and 0 otherwise by Definition 1. Considering $\mathbf{y} = \mathbf{f}(\mathbf{x})$ and $y = f(x)$, we obtain:

$$\Pr(\mathbf{Y} = \mathbf{y} | Y = y) \\ = \sum_{\substack{\mathbf{x} \in \mathrm{Sh}(x), \\ x, f(x) = y}} \Pr(\mathbf{Y} = \mathbf{f}(\mathbf{x}) | Y = f(x)) \Pr(\mathbf{X} = \mathbf{x}, X = x).$$

Using the equality

$$\Pr(\mathbf{X} = \mathbf{x}, X = x) = \Pr(\mathbf{X} = \mathbf{x} | X = x) \Pr(X = x)$$

and Definition 1, the second factor becomes $p' \Pr(X = x)$. The proof of Lemma 1 implies that $p' = |\mathcal{F}|^{-m(s_x-1)}$. Definition 4 implies that the first factor equals $|\mathcal{F}|^{m(s_x-1)-n(s_y-1)}$ for all $\mathbf{y} \in \mathrm{Sh}(y)$. We obtain

$$\Pr(\mathbf{Y} = \mathbf{y} | Y = y) \\ = \sum_{\substack{\mathbf{x} \in \mathrm{Sh}(x), \\ x, f(x) = y}} |\mathcal{F}|^{-n(s_y-1)} \Pr(X = x) = |\mathcal{F}|^{-n(s_y-1)}.$$

Hence, $\Pr(\mathbf{Y} = \mathbf{y} | Y = y)$ is equal to a constant $p = |\mathcal{F}|^{-n(s_y-1)}$ if $\mathbf{y} \in \mathrm{Sh}(y)$ and 0 otherwise satisfying a uniform masking. ■

Practice shows that adding the uniformity requirement to a non-completely shared function tends to blow up its mathematical complexity, as well as the cost of its implementation. In some applications it might be better to consider re-masking, for instance if random bits are available at low cost.

E. Uniformity for Cascaded and Parallel Functions

If the TI technique is used to protect cascaded functions, then extra measures like the ones discussed in the previous section need to be taken, such that the input for the following nonlinear operation is again a uniform masking. A similar situation occurs when the TI technique is used to protect several functional blocks acting in parallel on (partially) the same inputs. This occurs for example in implementations of the AES S-box using the tower field approach. If no special care is taken, then "local uniformity" of the distributions of the outputs of the individual blocks will not lead to "global uniformity" for the joint distributions of the outputs of all blocks. For example, let $\mathbf{f}, \mathbf{g}$ be two functions acting on the same uniform input $\mathbf{x}$. Then, even if $\mathbf{f}, \mathbf{g}$ are uniform shared functions, producing uniform $\mathbf{y} = \mathbf{f}(\mathbf{x})$ and $\mathbf{y}' = \mathbf{g}(\mathbf{x})$, this does not imply that $(\mathbf{y}, \mathbf{y}')$ is uniform. Like with cascaded functions, if each of the parallel blocks satisfies the properties of correctness and non-completeness, there will be no leakage of signals within the parallel blocks, but the lack of uniformity in the joint distribution of the output's masking can lead to information leakage if the outputs are combined as inputs to a following nonlinear function. At this time, the only known solution for this problem is re-masking.

F. Reducing the Randomness Used in a Re-masking Step

As mentioned in the beginning of Section II-D, we can generate a uniform masking from any share vector $\mathbf{X}$ by re-masking all its shares X_i using fresh random masks. Hence, we stress the following point.

Observation 1: A TI that uses re-masking does not need uniformly shared functions in order to resist first-order attacks.

However, re-masking all the shares of a masking can be a burden when generating fresh randomness is costly. The following theorem allows to reduce the amount of random bits used by re-masking steps of TIs: if $(X_1, \ldots, X_t)$ of a masking with s shares have a uniform distribution, only the remaining nonuniform fraction of the shares $(X_{t+1}, \ldots, X_s)$ needs to be re-masked.

Theorem 4: Let $(X_1, X_2, \ldots, X_s)$ be a masking of a (stochastic) variable $X \in \mathcal{F}^m$, where $\Pr(X_1 = x_1, \ldots, X_t = x_t) = |\mathcal{F}|^{-tm}, \forall (x_1, \ldots, x_t)$ for some t with $1 \leq t \leq s$.

Then the masking $(Y_1, \ldots, Y_s)$, defined by $Y_i = X_i$ for $1 \leq i \leq t$ and $Y_i = X_i + R_i$ for $t < i \leq s$, is a *uniform* masking of X, i.e.: $\Pr(Y_1 = y_1, Y_2 = y_2, \ldots, Y_s = y_s | X = y_1 + y_2 + \cdots y_s) = |\mathcal{F}|^{m(1-s)}$, provided that the R_i, $i = t+1, \ldots, s-1$ are independently and uniformly distributed random variables and that $R_s = -(R_{t+1} + \cdots + R_{s-1})$.

Proof: We give here a sketch of the proof. We have:

$$\begin{aligned} &\Pr(Y_1 = y_1, \ldots, Y_s = y_s | X = y_1 + y_2 + \cdots y_s) \\ &= \Pr(Y_1 = y_1, \ldots, Y_t = y_t | X = y_1 + y_2 + \cdots y_s) \\ &\quad \cdot \Pr(Y_{t+1} = y_{t+1}, \ldots, Y_s = y_s \hookleftarrow \\ &\quad \hookrightarrow | X = y_1 + \cdots y_s, Y_1 = y_1, \ldots, Y_t = y_t) \,. \end{aligned} \tag{5}$$

Since $Y_i = X_i$ for $1 \leq i \leq t$, the first factor equals $|\mathcal{F}|^{-tm}$. For the second factor we recall the definition of Y_{t+1} to obtain that:

$$\begin{aligned} &\Pr(Y_{t+1} = y_{t+1}) \\ &\quad = \sum_{x_{t+1}} \Pr(X_{t+1} = x_{t+1}) \underbrace{\Pr(R_{t+1} = y_{t+1} - x_{t+1})}_{|\mathcal{F}|^{-m}} \,. \end{aligned}$$

The same holds for $Y_{t+2}, \ldots, Y_{s-1}$ and since the R_i have independent distributions, we can equate the second factor of (5) to:

$$|\mathcal{F}|^{(1-s-t)m} \sum_{x_{t+1}, \ldots, x_{s-1}} \Pr(X_{t+1}{=}x_{t+1}, \ldots, X_{s-1}{=}x_{s-1}, Y_s{=}y_s | \hookleftarrow \hookrightarrow X{=}y_1{+}\cdots{+}y_s, X_1{=}x_1, \ldots, X_t{=}x_t) \,.$$

Recalling the definition of Y_s completes the proof. ∎

Clearly, the extra randomness required by the re-masking approach in some cases may be a worse problem than the blow-up in gate count caused by the uniform sharing approach.

G. Consequences

Theorem 1 and Theorem 2 can be proven using Definitions 1 and 3. Moreover Definition 3 is required only for the sake of the implementation's correctness. In contrast, if several circuits are cascaded (*pipelined*) or they run in parallel, the uniform sharing in Definition 4 is also needed in order to satisfy Definition 1 in the following circuit. However, using a uniform circuit can be avoided with (partial) re-masking. In other words, if we consider first-order DPA only, then there is no need to demand uniformity of a sharing that is followed by a re-masking step anyway. By relinquishing the uniformity requirement, it is often possible to reduce the number of shares and the size of the circuit as suggested in Theorem 4. This will be used in the next section in order to reduce the number of shares in the subcircuits of the AES S-box.

III. Implementation

In this section we will discuss three different TIs of AES which we refer to as raw, adjusted and nimble implementations. All implementations share the same data flow and timing. The implementations differ mostly in the S-box calculation and/or the number of shares that are used in different blocks of the algorithm. The *raw* implementation is from our paper at AfricaCrypt 2014 [17] and forms the basis of the other two implementations. Hence, we will mainly describe the raw implementation and point out the differences with the other two. The main feature of the raw implementation is that it uses the smallest possible number of shares for each function, except the linear transformations in the S-box, provided that the shared functions are uniform. In other words, all nonlinear operations are performed with $n > 2$ shares such that the circuits are uniform and n is as small as possible. The linear operations outside the S-box are performed with two shares, whereas the linear operations in the S-box use two, three or four shares (see Sect. III-B).

The *adjusted* implementation on the other hand ensures that at least three shares are used in every operation, including the linear ones. With this implementation we intend to observe the effect of moving from at least two shares to at least three shares in linear operations on the resistance against higher-order DPA, and to quantify the associated cost. In the *nimble* implementation the number of shares is always minimal, i.e. $n = d+1$ where d is the degree of the unshared function, even if the resulting shared function is not uniform. The uniformity of the circuit is satisfied by re-masking.

We will first describe the general data flow of our implementations in Section III-A. In Section III-B we will introduce different approaches to apply the TI to the AES S-box, which is the only nonlinear layer of the block cipher. We described the proposed designs in Verilog, separating component functions in modules, and verified their functionality with ModelSim. Then we used a standard tool chain to synthesize them using Synopsys Design Vision D-201-.03-SP4 with Faraday Standard Cell Library FSA0A_C_Generic_Core, which is based on UMC 0.18μm GenericII Logic Process with 1.8V voltage. We will conclude this section by providing the area, timing and randomness requirements of our designs in Section III-C. We look at the number of NAND gate equivalence (GE) for the area, represent the timing with clock cycles and calculate the randomness requirement in bits. We compare our implementations with the previous work in [16] which uses a similar standard cell library based on UMC 0.18μm logic process with 1.8V voltage.

A. General Data Flow

We use a serial implementation for round operations and key schedule as proposed in [16], [17] which requires only one S-box instance and loads the plaintext and key byte-wise in row-wise order. We also use one MixColumns instance that operates on the whole column and provides an output in one clock cycle. Due to this extreme serialization, one round requires at least 21 clock cycles even for the unprotected implementation [16]. All our TIs execute one round in 23 clock cycles. In the first 16 clock cycles, the plaintext is XORed with the key and sent to the S-box. Its output will be taken from the 3$^{\text{rd}}$ to the 18$^{\text{th}}$ clock cycles and stored in the state registers, i.e. the S-box is executed in three clock cycles. The ShiftRows operation is performed in the 19$^{\text{th}}$ clock cycle followed by four cycles of MixColumns calculation. The S-box takes its input from the key schedule for four cycles starting from the 18$^{\text{th}}$ cycle. In the 17$^{\text{th}}$, 22$^{\text{nd}}$ and 23$^{\text{rd}}$ clock cycles, the S-box inputs and unused

random bits are set to 0. Therefore, the calculation of AES takes $23 \times 10 + 16 = 246$ clock cycles, including 16 cycles to output the ciphertext.

1) Raw implementation: We use two sets of state registers, each consisting of sixteen 16-bit registers, corresponding to the two shares of the state. The MixColumns and the Key XOR operations are also performed with two shares. This can be seen in Fig. 1, as the key and the state registers are 256 bits implying the two shares.

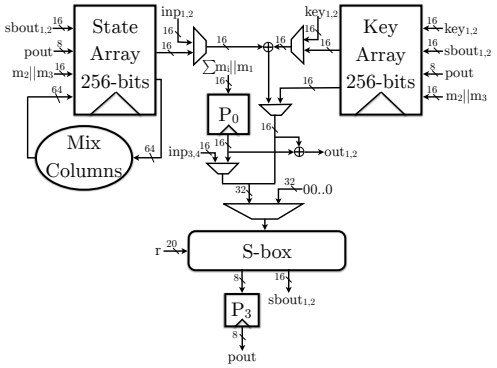

Figure 1: Architecture of the serialized TI of raw AES-128.

This TI of the S-box (details will be given in the following section) requires four input shares, therefore we initially share the plaintext in four shares. We share the key in two shares and XOR them with two of the plaintext shares before the S-box operation. More details about the key scheduling will be given later in this section. Besides the shared input, the S-box needs 20-bits of randomness r. The first two output shares $\text{sbout}_{1,2}$ are written to the state register S_{33} (Fig. 2) whereas the remaining share sbout_3 is written to register P_3. The data in the state registers are shifted to the left for the following 16 cycles so that the next output of the S-box can be stored in the same registers. During this shift, the data in P_3 (pout in Fig. 1) is XORed with the second share of the S-box output, which is in the state register S_{33}, to reduce the number of shares from three to two. To achieve this signal sig_2 is active from the 4^{th} to the 19^{th} clock cycle.

The ShiftRows operation is performed in the 19^{th} clock cycle with an irregular horizontal shift. In the next four clock-cycles, the data in the registers S_{00}, S_{10}, S_{20} and S_{30} are sent to the MixColumns operation, the rest of the registers are shifted to the left horizontally and the output of the MixColumns operation is written to the registers S_{03}, S_{13}, S_{23} and S_{33}. The MixColumns operation is implemented column-wise as in [16] and with two shares working in parallel. The registers except S_{10}, S_{11} and S_{12} are implemented as scan flip-flops (SFF) that are D-flip-flops (DFF) combined with 2-to-1 MUXes. They can operate with two inputs at reduced area cost. A single 2-to-1 MUX costs 3.33 GE and one bit register costs 5.33 GE whereas one bit SFF costs 6.33 GE in our library.

In the following AES rounds, we increase the number of shares of the S-box input from two to four, using 24 bits of randomness (three bytes each of which is referred to as m_i in the figures), one clock cycle before the S-box operation. To achieve this signal sig_1 is active for sixteen clock cycles, starting from the last clock-cycle of each round. We separate the increase of the number of shares and the nonlinear operation with registers to achieve the non-completeness property. The two additional shares are stored in P_0. The two shares in S_{00} are XORed with the two shares of the corresponding round key byte and sent to the S-box together with the two shares in P_0.

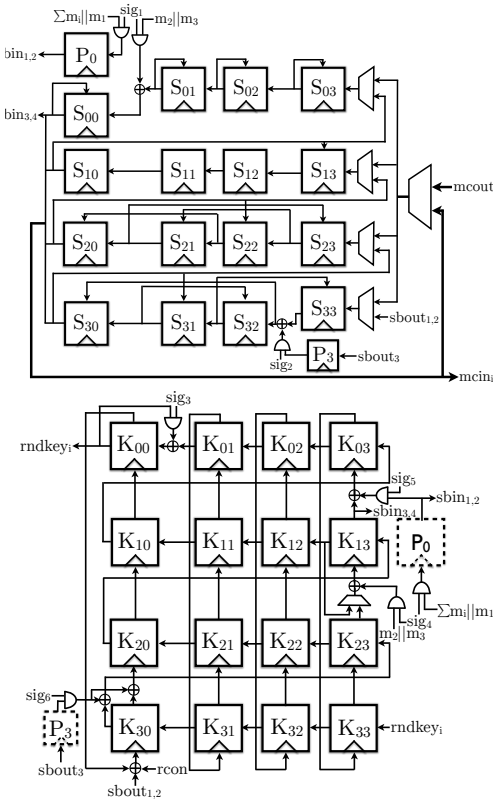

Figure 2: Architecture of the state (top) and key (bottom) arrays for our raw implementation where S_i, K_i and P_0 hold two shares and P_3 holds one share. The registers P_0 and P_3 are used by the state and the key array. The XOR of the value in P_3 and S_{33} (resp. K_{30}) is on one share of the value in register S_{33} (resp. K_{30}) whereas all the other combinational operations are on two shares.

The registers P_0 and P_3 are used for both the round transformations and the key scheduling.

Similar to the state array, the key array also consists of sixteen 16-bit registers, implemented as SFFs, each corresponding to the two shares of a byte in the key schedule. The round key is inserted from the register K_{33} in the first sixteen clock cycles of each round. For the next three clock cycles, the registers except the last column (K_{03}, K_{13}, K_{23} and K_{33}) are not clocked. The registers K_{03}, K_{23} and K_{33} are also not clocked in the 17^{th} clock cycle. In that clock cycle, we increase the number of shares in the register K_{13}. In the

following three clock cycles this re-sharing is done during the vertical shift from the register K_{23} to K_{13}, i.e. the re-sharing signal sig_4 is active from the 17^{th} to the 20^{th} clock cycle. Signal sig_5 is active from the 18^{th} to the 21^{st} clock cycle to reduce the number of shares back to two. The registers K_{03}, K_{13}, K_{23} and K_{33} are not clocked in the remaining two clock cycles of each round. We choose this way of irregular clocking to avoid using extra MUXes in our design. Two shares of the S-box output are XORed to the data in K_{00} in the last four clock cycles of each round. In the 20^{th} clock cycle the round counter $rcon$ is additionally XORed to one of these shares. The number of shares is reduced back to two by XORing the share in P_3 to one of the shares in K_{30}. Signal sig_3 is active in the first sixteen clock cycles except the 4^{th}, 8^{th}, 12^{th} and 16^{th} clock cycles. The round key is taken from the register K_{00} to be XORed with the corresponding plaintext before going to the S-box operation.

2) Adjusted implementation: This version works on three shares for both the state and the key schedule which increases the area significantly. The S-box still requires four input shares and outputs three shares, hence the register P_0 is reduced to 8-bits (one share) and the register P_3 is not required. Similar to the raw implementation, we use 24-bits of randomness to increase the number of shares from three to four one cycle before the S-box, i.e. each of the existing three shares is XORed with a random byte and the sum of these random bytes is taken as the fourth share. This also ensures uniformity of the S-box input. Together with the state, the number of shares for MixColumns and Key XOR increases to three.

3) Nimble implementation: Similar to the raw implementation, this one also uses two shares for the state and key arrays. The main difference is that the S-box needs three input shares instead of four. Hence the size of the register P_0 is reduced to 8-bits (one share). As a result, we need only 16-bits of randomness to increase the number of shares from two to three before the S-box operation, i.e. each share is XORed with one byte of randomness and the XOR of the random bytes is taken as the third share. The S-box requires 16-bits of extra randomness per iteration and outputs three shares. Hence the logic of the register P_3 to reduce the number of shares back to two stays the same.

B. TI of the AES S-box

The S-box implementations in [16] use the tower field approach up to GF(2^2) for a small implementation. Therefore, the only nonlinear operation is GF(2^2) multiplication which must be followed by registers and re-masking to avoid first order leakages.

We also chose to use the tower field approach, however, we decided to go until GF(2^4) instead of GF(2^2). With this approach, the GF(2^4) inverter (algebraic normal form provided in Appendix B) can be seen as a four bit permutation and the GF(2^4) multiplier (algebraic normal form provided in Appendix A) as a four bit multiplication both of which are well studied in [22]. Therefore, we can find uniform TIs for each of these nonlinear functions. This might allow us to reduce the number of fresh random bits needed since we will have fewer nonlinear blocks compared to [22] hence possibly require less re-masking in order to use their outputs. Moreover, with this approach the S-box calculation takes three clock cycles instead of five.

1) Raw implementation (Fig. 3): The uniformity of each function is individually satisfied. The uniform sharing with four input and three output shares that is used to share each term in the multiplication is provided in Appendix C. For the inversion, which belongs to class $\mathcal{C}^4_{282}$ [14], we consider two options. Either using four shares, which is the minimum number of shares necessary for a uniform implementation in that class, and decomposing the function into three uniform sub-functions as $Inv(x) = F(G(H(x)))$, or using five shares without any decomposition. Our experiments show that both versions have similar area requirements but need a different number of clock cycles. To reduce the number of cycles, we chose the version with five shares, generated by applying the formula in Appendix F to each term of the inversion. This sharing is found by using the method described in [9] which is slightly different from the *direct sharing* [14]. We chose this sharing since it can be implemented in hardware with less logic gates compared to the direct sharing.

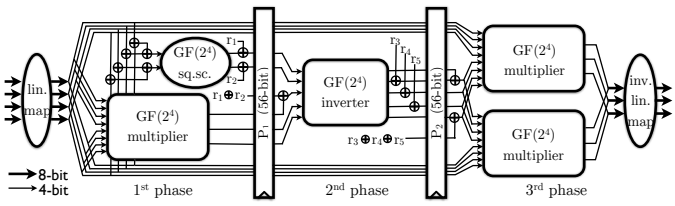

Figure 3: The S-box of the raw implementation.

Even though it is enough to use only two shares for linear operations, we sometimes chose to work on more than two shares to avoid the need of extra random bits. The linear map of the tower-field S-box operates on four shares since the multiplication needs four input shares. The inverter requires five input shares and the multiplication outputs only three shares, therefore we use two shares for the square scalar to have five shares in the beginning of the 2^{nd} phase. We use three shares for the inverse linear map of the tower-field S-box since the multiplication outputs three shares. For all the linear operations, the shared functions are created as instantiations of the unshared function for the first share and as unshared function without the constant term for the other shares.

During the combination of these uniform circuits, we face the challenges described in Section II-E to keep the uniformity in the pipeline registers. We apply re-masking on the first pipeline register where we combine the two output shares of the square scaler and the three output shares of the multiplier to generate five shares. Note that this combination also acts as the XOR of the outputs of the square scaler and the multiplier. By Theorem 4, it is enough to re-mask only the output shares of one of the functions to achieve uniformity. We choose to re-mask the output of the square scaler since it operates on less shares, hence requires less random bits. The correction mask,

i.e. the XOR of the masks, is XORed to one of the output shares of the multiplier to achieve correctness.

An other challenge is to satisfy the uniformity of the circuit as we increase or decrease the number of shares. This is achieved by introducing new masks before the S-box operation to increase from two to four shares and at the end of the 2nd phase to decrease from five to four shares. The output of the 3rd phase is not uniform when the three shares are considered together. However, we verified by simulation that each share individually is uniform, which implies that there is no first-order leakage in the following registers. We combine the first two shares with an XOR and keep the third share as it is to go back to two shares. We also verified that, after we decrease the number of shares to two, the output shares are uniform.

We always keep the XOR of the masks in the pipeline registers and complete the re-masking in the next clock cycle as in [16]. Overall, we need 44 fresh random bits per S-box operation including increasing the number of shares of the S-box input.

2) Adjusted implementation (Fig. 4): As mentioned in the earlier sections, the only difference between the raw and the adjusted implementation is that the adjusted implementation requires at least three shares for all the blocks including the linear operations in the S-box. For that reason, the shared square scaler circuit is instantiated with three shares. This S-box also requires 44-bits of randomness per iteration.

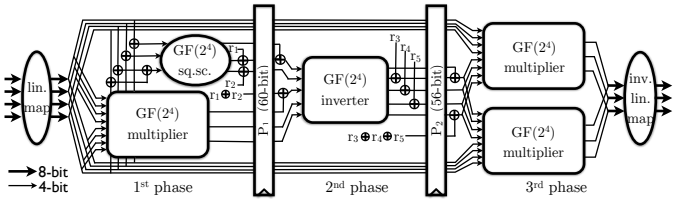

Figure 4: The S-box of the adjusted implementation.

3) Nimble implementation (Fig. 5): As can be observed in Figs. 3 and 4, we use fresh randomness at the end of the 1st phase to satisfy uniformity during the combination of the square scaler's and the multiplier's outputs, and after the inverter to break the dependency between the inputs of the multipliers in the 3rd phase. Since these re-masking steps conserve the uniformity property and the security of each block is achieved only by the correctness and non-completeness properties (Observation 1), we can discard the uniformity property and implement these nonlinear functions with the smallest number of shares n s.t. $n > d$, i.e. $n = d + 1$, where d is the degree of the unshared functions. We use the sharing with three input and output shares provided in Appendix D for each term of the multiplier and the sharing with four input and output shares provided in Appendix E for each term of the inverter. With this new construction, it is enough to have three input shares to the S-box since the multiplier block requires only three shares. We need to reduce the number of shares from five to four at the end of the 1st phase for the inverter and from four to three at the end of the 2nd phase for the following multipliers. This construction requires only 32-bits of extra randomness per S-box calculation, including increasing the number of shares for the S-box input.

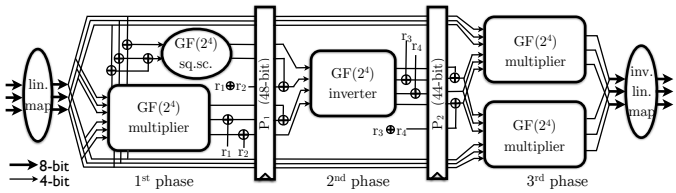

Figure 5: The S-box of the nimble implementation.

C. Performance

Like any other DPA countermeasure, TI also allows trade-offs between area, randomness and the resistance against DPA. In Table III, we provide the area costs (GE) and randomness requirements (bits) for the different S-box implementations. For all the implementations, we performed two different compilation methods. The first one is a regular compilation with the *compile* command, that does not optimize or merge modules, performed on the whole implementation. The second method on the other hand uses the *compile_ultra* command for each module to let the tool optimize each of them *individually* and combine the result. It is very important that the modules are not merged for area optimization in this step, to not violate the non-completeness property.

Table III: Synthesis results for different versions of S-box TI with *compile* / *compile_ultra* commands.

S-box	Raw	Adjusted	Nimble
Lin.Map.	168 / 120	168 / 120	126 / 90
Sq.Sc.	18	27	18
Multiplier	625 / 458	625 / 458	418 / 308
Inverter	618 / 490	618 / 490	594 / 375
Inv.Lin.Map	99 / 72	99 / 72	99 / 72
1st Ph.	919 / 704	916 / 701	646 / 500
2nd Ph.	690 / 562	702 / 574	654 / 435
3rd Ph.	1374 / 1013	1374 / 1013	959 / 713
Registers*	725	661	576
Total	3708 / 3004	3653 / 2949	2835 / 2224
Random.	44	44	32

*: including the registers P_0 and P_3

The total area results in Table III show that using nonuniformly shared functions as in the nimble implementation reduces the area cost significantly compared to the uniformly shared raw and adjusted implementations. This reduction is caused by the decreased number of shares used in the nonlinear blocks. Moreover, the required number of random bits per S-box also decreases together with the reduced number of shares since less shares need to be re-masked to satisfy uniformity.

In Table IV, we show the area, randomness requirements and timings of our AES implementations and compare them with the results in [16]. We again provide our results using the

same compilation techniques as the S-box implementations. The area costs for the state and the key arrays include the ANDs and XORs that are shown in Fig. 2. As expected in the raw and nimble implementations the cost of the state and key arrays together with the MixColumns are reduced by one third compared to [16] and the adjusted implementation, since we use two shares instead of three. All our versions have the same timing and use the same control module.

Table IV: Synthesis results for different versions of AES TI with *compile* / *compile_ultra* commands.

Design	[16]	Raw	Adjusted	Nimble
State Ar.	2529	1698	2473	1687
Key Ar.	2526	1890	2762	1844
S-box	4244	3708 / 3004	3653 / 2949	2835 / 2224
MixCol.	1120	770 / 544	1156 / 816	770 / 544
Control[1]	255	242	242	242
Key XOR	64	48	72	48
MUXes	376	746	853	693
Total	11114 / 11031	9102 / 8172	11221 / 10167	8119 / 7282
Cycles	266	246	246	246
Random.[2]	48	44	44	32

[1] including round constant and other [2] per S-box

In our implementations, the S-box occupies 30% to 40% of the total area. Compared to the implementation in [16] our S-boxes with uniform blocks are 13% smaller and our S-box with non-uniform blocks is 33% smaller. These results show a significant area and randomness improvement for the nimble implementation, indicating that using nonuniform shared functions can be advantageous if the uniformity of the circuit is satisfied by re-masking.

IV. Power Analysis

To evaluate the security of our designs in practice we implement them on a SASEBO-G board [23] using Xilinx ISE version 10.1. The Verilog descriptions of the designs are the same as for the ASIC evaluations, but we replaced all SFFs by DFFs and MUXes because SFFs are not available. We use the "keep hierarchy" constraint to prevent the tools from optimizing over module boundaries (see the last paragraph of Sect. II-B and the last sentence before Table III). Apart from that we use the standard tool chain. The board features two Xilinx Virtex-II Pro FPGA devices: we implement the TI AES and a pseudorandom number generator (PRNG) on the crypto FPGA (xc2vp7) while the control FPGA (xc2vp30) handles I/O with the measurement PC and other equipment. The PRNG that generates all random bits is implemented as AES-128 in CTR mode.

We measure the power consumption of the crypto FPGA during the first 1.5 rounds of TI AES as the voltage drop over a 1Ω resistor in the FPGA core GND line. The output of the passive probe is sampled with a Tektronix DPO 7254C digital oscilloscope at 1GS/s sampling rate.

A. Methodology

We define two main goals for our practical evaluations. First, we want to verify our implementations' resistance against first-order attacks. But in practice adversaries are of course not restricted to applying such attacks. Therefore, our second goal is to assess the level of security our implementations provide against other, e.g. higher-order, power analysis attacks.

Since there is no single, all-embracing test to evaluate the security of an implementation, we test its resistance against state-of-the-art attacks. We narrow the evaluation to univariate attacks because our implementations process all shares of a value in parallel. Estimating the information-theoretic metric by Standaert et al. [24] is out of reach. It would require estimation of at least 2^{48} Gaussian templates.

We make several choices that are in favor of an adversary and make attacks easier. First, to minimize algorithmic noise the PRNG and the TI AES do not operate in parallel, i.e. the PRNG generates and stores a sufficient number of random bits before each TI AES operation. In practice, running them in parallel will increase the level of noise and thus the number of measurements needed for an attack to succeed. Second, we provide the crypto FPGA with a stable 3MHz clock frequency to ensure that the traces are well aligned and the power peaks of adjacent clock cycles do not overlap (this would also help to assign a possibly identified leak to a specific clock cycle). In practice, clocking the device at a faster or unstable clock will make attacks harder. Third, we let the adversary know the implementation. Specifically, if the PRNG was switched off the adversary would be able to correctly compute bit values and bit flips under the correct key hypothesis. In practice, obscurity is often used as an additional layer of security. Fourth, we use synchronous (over-)sampling [25] to avoid clock drift and achieve the best possible alignment. In practice, secure devices use an internal (and unstable) clock source which prevents synchronous sampling and increases the number of measurements needed for an attack to succeed.

B. PRNG switched off

To confirm that our setup works correctly and to get some reference values we first attack the implementations with the PRNG switched off. We expect that the implementations can be broken with many first-order attacks. As example, we applied correlation DPA attacks [26] that use the Hamming distance of two consecutive S-box outputs as power model. The attacks require $2 \cdot 2^8$ key hypotheses. To reduce the computational complexity we let the adversary know one key byte and aim to recover the second one.

Since the adversary knows the implementation, he can choose to compute the Hamming distance over three 8-bit registers (all versions; S_{33} and P_3; output of the S-box in three shares), two 8-bit registers (raw and nimble; S_{32}; one cycle later; two shares) or ignore the details and compute the distance over a single 8-bit register as if it was a plain implementation. For all versions, only a few hundred traces are required to recover the key with any of these attacks. It is worth noting that the highest correlation peaks do not occur at the S-box output registers, but three resp. two clock cycles later when the same bit-flips occur in register S_{30}. This register drives the MixColumns logic and therefore has a much greater fanout.

We also applied correlation collision attacks [7] that target combinational logic. The attacks compute two sets of mean traces for the values of two processed plaintext bytes and shift the mean traces in the time domain to align them. They aim to recover the linear difference between the two key bytes involved. To do so, they permute one set of mean traces according to a hypothesis on the linear difference and then correlate both sets of mean traces. The results show that this attack is successful with a few thousand measurements for all versions. For more details and figures regarding the raw implementation, see [17].

C. PRNG switched on

Next we repeat the evaluation with the PRNG switched on, i.e. the TI AES uses unknown and unpredictable random bits. For the DPA attacks using the Hamming distance over two or three registers as power model, we suppose these bits were zero.

1) Raw implementation: Fig. 6 shows the results of the first-order attacks against the protected implementation using 10 million measurements. The results show that the attacks fail.

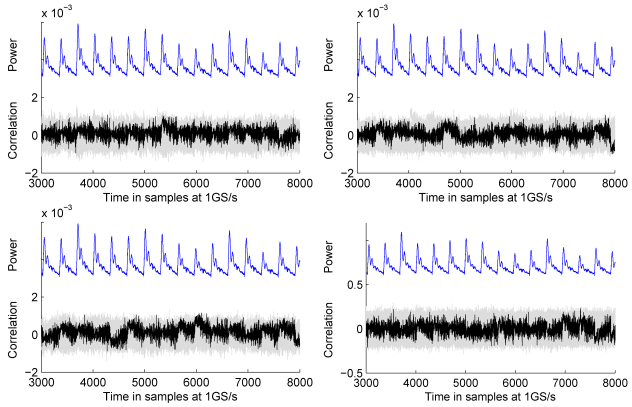

Figure 6: Results of first-order DPA and correlation collision attacks on raw implementation with PRNG on computed using 10 million traces; top, left: HD over 1 register; top, right: HD over 2 registers; bottom, left: HD over 3 registers; bottom, right: correlation collision.

We proceed with higher-order attacks to assess the level of security this implementation provides. For our second-order DPA attacks we use the same power models as before but center and then square the traces (for each time sample) before correlating [2], [27], [28]. Second-order correlation collision attacks work as above with mean traces replaced by variance traces [13].

Fig. 7 (top, left) shows the results of the second-order DPA attack that uses the Hamming distance in a single register as power model (as if it was a plain implementation) using 10 million measurements. We note that the highest correlation peak occurs again when the same bitflips happen in register S_{30}, similar to when the PRNG was switched off. The attack requires about 600 000 traces to succeed, as shown in Fig. 7 (top, right). Second-order DPA attacks using the Hamming distance over two resp. three registers as power model failed to recover the key, presumably because we do not know the masks' values and assume they are zero.

Fig. 7 (bottom, left) shows the results of the second-order correlation collision attack using 10 million measurements. The attack requires about 3.5 million traces to succeed, as shown in Fig. 7 (bottom, right).

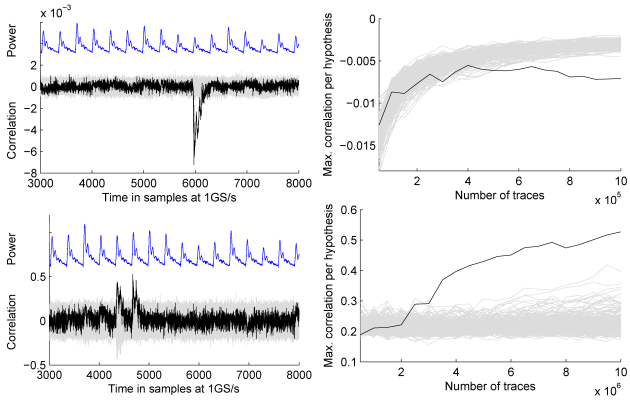

Figure 7: Results of second-order DPA (top) and correlation collision (bottom) attacks on raw implementation with PRNG on computed using 10 million traces; right: min./max. correlation coefficient per hypothesis (from the overall time span) over number of traces used.

2) Adjusted implementation: We performed the same analysis as on the raw implementation. Fig. 8 shows that neither the first-order DPA attack that uses the Hamming distance in one register as power model nor the first-order correlation collision attack work with 10 million traces, as expected.

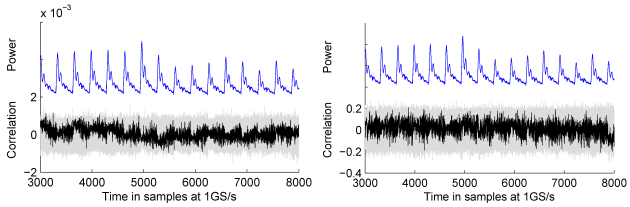

Figure 8: Results of first-order DPA (left) and correlation collision (right) attacks on adjusted implementation with PRNG on computed using 10 million traces.

Unlike our result for the raw implementation, we observe that second-order DPA does not work even with 10 million traces as shown in Fig. 9 (top). This result is natural since the adjusted implementation uses three shares instead of two in register S_{33} (and the entire state array). We expect a third-order DPA attack that exploits the third standardized moment of the traces to be possible, however the available 10 million traces were not enough.

On the other hand, a second-order correlation collision attack still succeeds, indicating leakage from possible glitches in the S-box, as shown in Fig. 9 (bottom, left). Recall that the adjusted implementation uses at least three shares in every operation. Compared to Fig. 7 (bottom, left) the second correlation peak does not show. This might be the reason why the attack becomes harder, as shown in Fig. 9 (bottom, right). It is successful with about 4 million traces, but the separation of the correct key from the wrong keys is poor, even using 10 million traces. This observation indicates that the leakage that leads to the first correlation peak is almost linear and therefore harder to exploit.

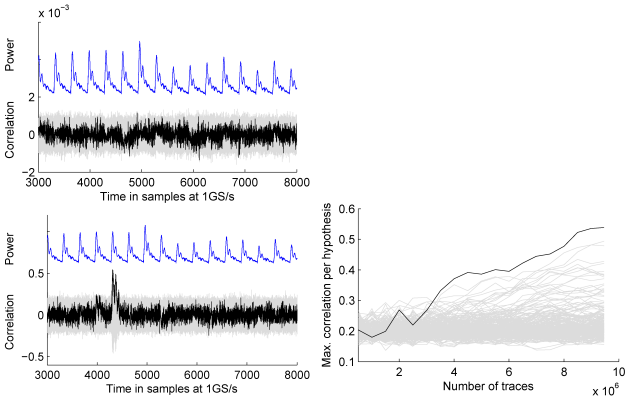

Figure 9: Results of second-order DPA (top) and correlation collision (bottom) attacks on adjusted implementation with PRNG on computed using 10 million traces; right: min./max. correlation coefficient per hypothesis (from the overall time span) over number of traces used.

3) Nimble implementation: We performed the same analysis as on the raw implementation and the results are similar. First-order DPA and correlation collision attacks fail with 10 million traces. Both second-order DPA and correlation collision attacks show peaks (Fig. 10, left) in the same clock cycle as for the raw implementation. They succeed with about 600 000 and 8.5 million traces, respectively, as shown in Fig. 10 (right). However, we observe that the correlation collision attack requires more traces to be successful than for the raw implementation. We suspect that this is due to the simpler component functions of the nimble implementation, which cause less glitches in the circuit.

D. Discussion

The first goal of our evaluation is to verify our implementations' resistance against first-order attacks. But this goal is always limited by the number of measurements at hand. It is simply not possible to demonstrate resistance against attacks with an infinite number of traces. We have shown that our implementations resist state-of-the-art first-order attacks with 10 million traces in conditions that are strongly in favor of the adversary (no algorithmic noise from the PRNG, knowledge of the implementation, slow and stable clock, best possible

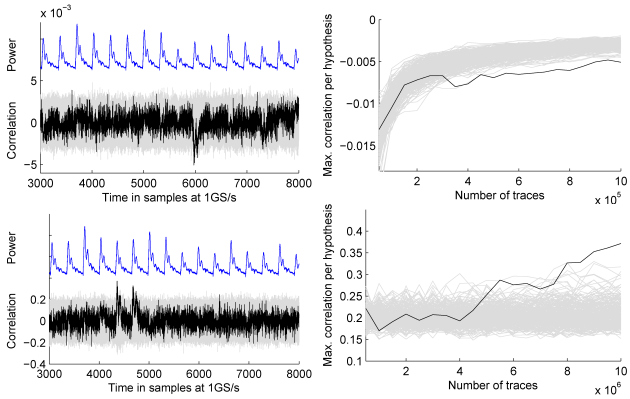

Figure 10: Results of second-order DPA (top) and correlation collision (bottom) attacks on nimble implementation with PRNG on computed using 1 million and 10 million traces, respectively; right: min./max. correlation coefficient per hypothesis (from the overall time span) over number of traces used.

alignment). Given the theoretical foundations of TI and the correctness of our implementations, we are convinced that our implementations resist first-order attacks with any number of measurements, but we have no way to demonstrate that.

The second goal of our evaluation is to assess the level of security our implementations provide against higher-order attacks and to relate the results to the area and randomness requirements. In the same adversary-friendly conditions, the most trace-efficient second-order attack in our evaluation requires about 600 000 traces for the raw and the nimble implementations. The attack exploits that the state array is in two shares, which is common to both implementations that mainly differ in the S-box implementation. Since the nimble implementation requires less resources and provides a similar level of security, it is preferable over the raw implementation.

As expected, the adjusted implementation with at least three shares in all operations provides better security than the raw implementation it is based on. The same second-order DPA attack that succeeded with 600 000 traces against the raw implementation fails against the adjusted implementation even with 10 million traces. Also a third-order DPA attack against the adjusted implementation fails with 10 million measurements. The trace requirement for a successful second-order correlation collision attack increases only slightly from 3.5 million to about 4 million, but the separation of the correct key from the wrong keys is much poorer. The price of this increase in security is a roughly 23% larger circuit (randomness requirements and timings are identical).

V. Conclusion

We discuss three different versions of TIs of AES. We show that it is possible to achieve first-order DPA resistance with non-uniform shared functions if re-masking is applied properly. In the case of AES, our "non-uniform" nimble implementation

requires less randomness than our "uniform" raw implementation, due to the decreased number of shares. However, for other algorithms and other S-boxes, re-masking may increase the amount of randomness required. This idea can be used to trade-off between the randomness and area requirements. Moreover, we empirically confirm that increasing the number of shares has a significant impact on the performance of higher-order attacks, which provides another trade-off between area and DPA resistance. Our most efficient implementation is approximately 8k GE small and requires only 32 bits of fresh randomness per S-box calculation, which is a significant improvement over all previous works.

REFERENCES

[1] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *CRYPTO*, ser. LNCS, vol. 1666. Springer, 1999, pp. 388–397.

[2] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *CRYPTO*, ser. LNCS, vol. 1666. Springer, 1999, pp. 398–412.

[3] L. Goubin and J. Patarin, "DES and differential power analysis the "duplication" method," in *CHES*, ser. LNCS, vol. 1717. Springer, 1999, pp. 158–172.

[4] T. S. Messerges, "Securing the AES finalists against power analysis attacks," in *FSE*, ser. LNCS, vol. 1978. Springer, 2000, pp. 150–164.

[5] S. Mangard, T. Popp, and B. M. Gammel, "Side-channel leakage of masked CMOS gates," in *CT-RSA*, ser. LNCS, vol. 3376. Springer, 2005, pp. 351–365.

[6] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully attacking masked AES hardware implementations," in *CHES*, ser. LNCS, vol. 3659. Springer, 2005, pp. 157–171.

[7] A. Moradi, O. Mischke, and T. Eisenbarth, "Correlation-enhanced power analysis collision attack," in *CHES*, ser. LNCS, vol. 6225. Springer, 2010, pp. 125–139.

[8] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold implementations against side-channel attacks and glitches," in *ICICS*, ser. LNCS, vol. 4307. Springer, 2006, pp. 529–545.

[9] S. Nikova, V. Rijmen, and M. Schläffer, "Secure hardware implementation of nonlinear functions in the presence of glitches," *J. Cryptology*, vol. 24, no. 2, pp. 292–321, 2011.

[10] E. Prouff and T. Roche, "Higher-order glitches free implementation of the AES using secure multi-party computation protocols," in *CHES*, ser. LNCS, vol. 6917. Springer, 2011, pp. 63–78.

[11] A. Moradi and O. Mischke, "On the simplicity of converting leakages from multivariate to univariate - (case study of a glitch-resistant masking scheme)," in *CHES*, ser. LNCS, vol. 8086. Springer, 2013, pp. 1–20.

[12] L. Batina, B. Gierlichs, E. Prouff, M. Rivain, F.-X. Standaert, and N. Veyrat-Charvillon, "Mutual Information Analysis: a Comprehensive Study," *J. Cryptol.*, vol. 24, no. 2, pp. 269–291, Apr. 2011.

[13] A. Moradi, "Statistical tools flavor side-channel collision attacks," in *EUROCRYPT*, ser. LNCS, vol. 7237. Springer, 2012, pp. 428–445.

[14] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz, "Threshold implementations of all 3×3 and 4×4 S-boxes," in *CHES*, ser. LNCS, vol. 7428. Springer, 2012, pp. 76–91.

[15] A. Poschmann, A. Moradi, K. Khoo, C.-W. Lim, H. Wang, and S. Ling, "Side-channel resistant crypto for less than 2300 GE," *J. Cryptology*, vol. 24, no. 2, pp. 322–345, 2011.

[16] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the limits: A very compact and a threshold implementation of AES," in *EUROCRYPT*, ser. LNCS, vol. 6632. Springer, 2011, pp. 69–88.

[17] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "A more efficient AES threshold implementation," in *AFRICACRYPT*, ser. LNCS. Springer, 2014, vol. 8469, pp. 267–284.

[18] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Building power analysis resistant implementations of KECCAK," Second SHA-3 candidate conference, August 2010.

[19] B. Bilgin, A. Bogdanov, M. Kneževic, F. Mendel, and Q. Wang, "Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware," in *CHES*, ser. LNCS. Springer, 2013, vol. 8086, pp. 142–158.

[20] D. Canright, "A very compact S-box for AES," in *CHES*, ser. LNCS, vol. 3659. Springer, 2005, pp. 441–455.

[21] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "KECCAK specifications," NIST SHA-3 contest 2008.

[22] B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz, "Threshold implementations of all 3×3 and 4×4 S-boxes," Cryptology ePrint Archive, Report 2012/300, 2012, http://eprint.iacr.org/.

[23] AIST, "Side-channel Attack Standard Evaluation BOard," http://staff.aist.go.jp/akashi.satoh/SASEBO/en/.

[24] F.-X. Standaert, T. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *EUROCRYPT*, ser. LNCS, vol. 5479. Springer, 2009, pp. 443–461.

[25] T. S. Messerges, "Power analysis attacks and countermeasures on cryptographic algorithms," Ph.D. dissertation, University of Illinois at Chicago, 2000.

[26] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *CHES*, ser. LNCS, vol. 3156. Springer, 2004, pp. 16–29.

[27] E. Prouff, M. Rivain, and R. Bevan, "Statistical analysis of second order differential power analysis," *IEEE Trans. Computers*, vol. 58, no. 6, pp. 799–811, 2009.

[28] J. Waddle and D. Wagner, "Towards efficient second-order power analysis," in *CHES*, ser. LNCS, M. Joye and J.-J. Quisquater, Eds., vol. 3156. Springer, pp. 1–15.

APPENDIX

A. Multiplier in $GF(2^4)$

$(Y_1, Y_2, Y_3, Y_4) = (X_1, X_2, X_3, X_4) \times (X_5, X_6, X_7, X_8)$

$$
\begin{aligned}
Y_1 &= X_1X_5 \oplus X_3X_5 \oplus X_4X_5 \oplus X_2X_6 \oplus X_3X_6 \oplus X_1X_7 \oplus X_2X_7 \oplus X_3X_7 \oplus X_4X_7 \\
&\quad \oplus X_1X_8 \oplus X_3X_8 \\
Y_2 &= X_2X_5 \oplus X_3X_5 \oplus X_1X_6 \oplus X_2X_6 \oplus X_4X_6 \oplus X_1X_7 \oplus X_3X_7 \oplus X_2X_8 \oplus X_4X_8 \\
Y_3 &= X_1X_5 \oplus X_2X_5 \oplus X_3X_5 \oplus X_4X_5 \oplus X_1X_6 \oplus X_3X_6 \oplus X_1X_7 \oplus X_2X_7 \oplus X_3X_7 \oplus \\
&\quad X_1X_8 \oplus X_4X_8 \\
Y_4 &= X_1X_5 \oplus X_3X_5 \oplus X_2X_6 \oplus X_4X_6 \oplus X_1X_7 \oplus X_4X_7 \oplus X_2X_8 \oplus X_3X_8 \oplus X_4X_8
\end{aligned}
$$

B. Inverter in $GF(2^4)$

$(Y_1, Y_2, Y_3, Y_4) = Inv(X_1, X_2, X_3, X_4)$

$$
\begin{aligned}
Y_1 &= X_3 \oplus X_4 \oplus X_1X_3 \oplus X_2X_3 \oplus X_2X_3X_4 \\
Y_2 &= X_4 \oplus X_1X_3 \oplus X_2X_3 \oplus X_2X_4 \oplus X_1X_3X_4 \\
Y_3 &= X_1 \oplus X_2 \oplus X_1X_3 \oplus X_1X_4 \oplus X_1X_2X_4 \\
Y_4 &= X_2 \oplus X_1X_3 \oplus X_1X_4 \oplus X_2X_4 \oplus X_1X_2X_3
\end{aligned}
$$

C. Sharing with 4 Input 3 Output Shares

$$
\begin{aligned}
F &= XY, \quad \text{where} \\
F &= F_1 \oplus F_2 \oplus F_3 \\
X &= X_1 \oplus X_2 \oplus X_3 \oplus X_4 \\
Y &= Y_1 \oplus Y_2 \oplus Y_3 \oplus Y_4
\end{aligned}
$$

$$
\begin{aligned}
F_1 &= (X_2 \oplus X_3 \oplus X_4)(Y_2 \oplus Y_3) \oplus Y_4 \\
F_2 &= ((X_1 \oplus X_3)(Y_1 \oplus Y_4)) \oplus X_1Y_3 \oplus X_4 \\
F_3 &= ((X_2 \oplus X_4)(Y_1 \oplus Y_4)) \oplus X_1Y_2 \oplus X_4 \oplus Y_4
\end{aligned}
$$

D. Sharing with 3 Input 3 Output Shares

$$
\begin{aligned}
F &= XY \oplus Z, \quad \text{where} \\
F &= F_1 \oplus F_2 \oplus F_3 \\
X &= X_1 \oplus X_2 \oplus X_3 \\
Y &= Y_1 \oplus Y_2 \oplus Y_3 \\
Z &= Z_1 \oplus Z_2 \oplus Z_3
\end{aligned}
$$

$$
\begin{aligned}
F_1 &= ((X_2 \oplus X_3)(Y_2 \oplus Y_3)) \oplus Z_2 \\
F_2 &= (X_1Y_3 \oplus Y_1X_3 \oplus X_1Y_1) \oplus Z_3 \\
F_3 &= (X_1Y_2 \oplus Y_1X_2) \oplus Z_1
\end{aligned}
$$

E. Sharing with 4 Input 4 Output Shares

$$
\begin{aligned}
F &= XYZ \oplus XY \oplus Z, \quad \text{where} \\
F &= F_1 \oplus F_2 \oplus F_3 \oplus F_4 \\
X &= X_1 \oplus X_2 \oplus X_3 \oplus X_4 \\
Y &= Y_1 \oplus Y_2 \oplus Y_3 \oplus Y_4 \\
Z &= Z_1 \oplus Z_2 \oplus Z_3 \oplus Z_4
\end{aligned}
$$

$$
\begin{aligned}
F_1 &= ((X_2 \oplus X_3 \oplus X_4)(Y_2 \oplus Y_3 \oplus Y_4)(Z_2 \oplus Z_3 \oplus Z_4)) \\
&\quad \oplus ((X_2 \oplus X_3 \oplus X_4)(Y_2 \oplus Y_3 \oplus Y_4)) \oplus Z_2 \\
F_2 &= (X_1(Y_3 \oplus Y_4)(Z_3 \oplus Z_4) \oplus Y_1(X_3 \oplus X_4)(Z_3 \oplus Z_4) \oplus Z_1(X_3 \oplus X_4)(Y_3 \oplus Y_4) \\
&\quad \oplus X_1Y_1(Z_3 \oplus Z_4) \oplus X_1Z_1(Y_3 \oplus Y_4) \oplus Y_1Z_1(X_3 \oplus X_4) \oplus X_1Y_1Z_1) \\
&\quad \oplus (X_1(Y_3 \oplus Y_4) \oplus Y_1(X_3 \oplus X_4) \oplus X_1Y_1) \oplus Z_3 \\
F_3 &= (X_1Y_1Z_2 \oplus X_1Y_2Z_1 \oplus X_2Y_1X_1 \oplus X_1Y_2Z_2 \oplus X_2Y_1Z_2 \oplus X_2Y_2Z_1 \oplus X_1Y_2Z_4 \\
&\quad \oplus X_2Y_1Z_4 \oplus X_1Y_4Z_2 \oplus X_2Y_4Z_1 \oplus X_4Y_1Z_2 \oplus X_4Y_2Z_1) \oplus (X_1Y_2 \oplus Y_1X_2) \oplus Z_4 \\
F_4 &= (X_1Y_2Z_3 \oplus X_1Y_3Z_2 \oplus X_2Y_1Z_3 \oplus X_2Y_3Z_1 \oplus X_3Y_1Z_2 \oplus X_3Y_2Z_1) \oplus 0 \oplus Z_1
\end{aligned}
$$

F. Sharing with 5 Input 5 Output Shares

$$
\begin{aligned}
F &= XYZ \oplus XY \oplus Z, \quad \text{where} \\
F &= F_1 \oplus F_2 \oplus F_3 \oplus F_4 \oplus F_5 \\
X &= X_1 \oplus X_2 \oplus X_3 \oplus X_4 \oplus X_5 \\
Y &= Y_1 \oplus Y_2 \oplus Y_3 \oplus Y_4 \oplus Y_5 \\
Z &= Z_1 \oplus Z_2 \oplus Z_3 \oplus Z_4 \oplus Z_5
\end{aligned}
$$

$$
\begin{aligned}
F_1 &= ((X_2 \oplus X_3 \oplus X_4 \oplus X_5)(Y_2 \oplus Y_3 \oplus Y_4 \oplus Y_5)(Z_2 \oplus Z_3 \oplus Z_4 \oplus Z_5)) \\
&\quad \oplus ((X_2 \oplus X_3 \oplus X_4 \oplus X_5)(Y_2 \oplus Y_3 \oplus Y_4 \oplus Y_5)) \oplus Z_2 \\
F_2 &= (X_1(Y_3 \oplus Y_4 \oplus Y_5)(Z_3 \oplus Z_4 \oplus Z_5) \oplus Y_1(X_3 \oplus X_4 \oplus X_5)(Z_3 \oplus Z_4 \oplus Z_5) \\
&\quad \oplus Z_1(X_3 \oplus X_4 \oplus X_5)(Y_3 \oplus Y_4 \oplus Y_5) \oplus X_1Y_1(Z_3 \oplus Z_4 \oplus Z_5) \oplus X_1Z_1(Y_3 \oplus Y_4 \oplus Y_5) \\
&\quad \oplus Y_1Z_1(X_3 \oplus X_4 \oplus X_5) \oplus X_1Y_1Z_1) \oplus (X_1(Y_3 \oplus Y_4 \oplus Y_5) \oplus Y_1(X_3 \oplus X_4 \oplus X_5) \\
&\quad \oplus X_1Y_1) \oplus Z_3 \\
F_3 &= (X_1Y_1Z_2 \oplus X_1Y_2Z_1 \oplus X_2Y_1X_1 \oplus X_1Y_2Z_2 \oplus X_2Y_1Z_2 \oplus X_2Y_2Z_1 \oplus X_1Y_2Z_4 \\
&\quad \oplus X_2Y_1Z_4 \oplus X_1Y_4Z_2 \oplus X_2Y_4Z_1 \oplus X_4Y_1Z_2 \oplus X_4Y_2Z_1 \oplus X_1Y_2Z_5 \oplus X_2Y_1Z_5 \\
&\quad \oplus X_1Y_5Z_2 \oplus X_2Y_5Z_1 \oplus X_5Y_1Z_2 \oplus X_5Y_2Z_1) \oplus (X_1Y_2 \oplus Y_1X_2) \oplus Z_4 \\
F_4 &= (X_1Y_2Z_3 \oplus X_1Y_3Z_2 \oplus X_2Y_1Z_3 \oplus X_2Y_3Z_1 \oplus X_3Y_1Z_2 \oplus X_3Y_2Z_1) \oplus 0 \oplus Z_5 \\
F_5 &= 0 \oplus 0 \oplus Z_1
\end{aligned}
$$

Svetla Nikova Svetla Nikova got her Master degree in mathematics from Sofia University (Bulgaria), and her Ph.D. degree from Eindhoven University of Technology (The Netherlands). From 1999 till 2008 she was a postdoctoral researcher in ESAT/COSIC, KU Leuven. From 2008 till 2012 she worked as an assistant professor in the University of Twente. Since March 2012, she is a research expert at the research group COSIC, KU Leuven. Svetla Nikova's research expertise is in cryptography, in particular lightweight cryptographic algorithms, side-channel resistant implementations, symmetric crypto primitives.

Ventzislav Nikov Ventzislav Nikov got a Master degree in mathematics from Sofia University Št. Kliment Ochridskiĭn 1992 and a Ph.D. in Cryptography from Eindhoven University of Technology in 2005. He worked as Security Expert & Architect at ACUNIA (2000 - 2002) and Philips (2004-2007). Since 2007 he has joined NXP Semiconductors as Principal Security Researcher and Architect. His research interests are in cryptography (e.g. side-channel resistant implementations, symmetric crypto primitives, etc.), security applications (e.g. content protection, software/hardware IP protection, etc.) and secure computations.

Vincent Rijmen Vincent Rijmen (senior member) is full professor at KU Leuven, Dept. ESAT/IBBT (Belgium). He is co-designer of the Advanced Encryption Standard (AES) and the ISO/IEC standard hash function Whirlpool. His research interests include cryptanalysis of ciphers and cryptographic hash functions, as well as mathematical countermeasures against side-channel attacks.

Begül Bilgin Begül Bilgin received her Master degree in Cryptography from Middle East Technical University (Turkey) in 2010. Since 2011, she is a Ph.D. student in KU Leuven and the University of Twente (The Netherlands). Her research interests include lightweight symmetric-key cryptography, implementations attacks and their countermeasures.

Benedikt Gierlichs Benedikt Gierlichs received the Ph.D. degree in electrical engineering from KU Leuven (Belgium) in 2011. He is since then post-doctoral researcher at the COSIC research group of KU Leuven and post-doctoral fellow with the Research Foundation Flanders (Belgium). His research interests include applied cryptography, physical security of real-time embedded systems for security and cryptography, implementation attacks, countermeasures and design methodologies for security.